Trabajando con precisión en números decimales en Python

Yan Orestes
29/09/2020

Es muy posible que ya hayas encontrado pequeños errores de precisión con números float, esto me pasó a mí cuando trabajé en una aplicación en **Python** para controlar los gastos y ganancias de la empresa donde trabajo.

Al principio, es sencillo, simplemente guardo los valores en variables y luego los paso a una base de datos propia. En julio, tuvimos 5 ventas de $ `99.91` y compramos 3 equipos de $ `110.10`:

```
ganancias_julio = 99.91 * 5
gastos_julio = 110.1 * 3

almacena_en_la_base (ganancias_julio, gastos_julio)
```

A partir de eso, los expertos en finanzas de la compañía realizan análisis para intentar mejorar nuestros resultados. Mi código es muy simple, pero tiene una lógica clara que debería funcionar bien. A pesar de eso, a fin de mes el jefe terminó retándome. Esto se debe a que algunos cálculos no coincidieron con los resultados reales que tuvimos. Estaba confundido y fui a probar mi código para verificar si todo está como debería:

```
ganancias_julio = 99.91 * 5
print (ganancias_julio)

gastos_julio = 110.1 * 3
print (gastos_julio)
```

Mira los resultados que obtuve:

```
499.54999999999995
330.29999999999995
```

Espera... ¿qué? Haciendo las cuentas manualmente, o en la calculadora, los resultados son diferentes: 499.55 y 330.3. ¿Por qué Python me entregó estos números largos y, sobre todo, **incorrectos**?

El problema del float

La lógica de nuestras cuentas está correcta, el problema, como hemos visto, está en los mismos resultados que nos da Python. Python no sabe hacerlas bien, ¿sería eso?

Bueno, la respuesta puede ser sí ... ¡y no! En primer lugar, esta pregunta no es exclusiva de Python, sino de la computación y de cómo se ocupa de los números de **punto flotante (nuestro querido float)**. Además, no es exactamente un problema; ¡vamos a entender!

En el nivel más bajo, las computadoras funcionan con la diferencia en **dos** estados eléctricos: bajo y alto voltaje, encendido y apagado, verdadero y falso. Por lo tanto, tenemos el **binario**, el famoso **0 y 1**, y es gracias a este sistema que las computadoras pueden ser tan rápidas con algunas cosas.

Sin embargo, al utilizar el formato binario para los números de punto flotante, las computadoras no pueden representar con precisión exacta algunas fracciones (como 0.98 y 0.1). De esta manera, estos números se redondean automáticamente al más cercano que se ajuste a la posibilidad del binario, lo que resulta en un pequeño error de precisión.

En general, este error es demasiado pequeño para ser considerado relevante, pero hay situaciones en las que no podemos ignorarlo, ¡como ahora!

En nuestro caso, como estamos tratando con dinero, necesitamos una **mayor precisión**. Hemos visto cómo redondear y formatear valores monetarios antes, pero incluso esta

forma puede dar lugar a pequeños errores de redondeo que queremos evitar. ¿Y ahora?

Trabajando con enteros

Mientras trabajamos con dinero, podemos pensar rápidamente en una alternativa que evitaría el problema de los números de punto flotante, trabajando solo con **números enteros**. Pero ¿cómo? Sabemos que **1 dólar equivale a 100 centavos**, entonces simplemente podemos trabajar con esta subunidad y evitar números decimales.

En nuestro caso, tuvimos $ `188.98` y $ `1113.10` que también se puede representar, respectivamente, por 18898¢ y 1310¢ - ¡dos números enteros!

Como estamos trabajando con Python, aún evitamos (al principio) el problema de overflow de número entero estándar, que es muy limitado en algunos lenguajes (como en Java, que solo llega a **2.147.483.647**).

Dado que la base de datos está guardando números enteros, ahora podemos simplemente dividir por **100** al imprimir los valores, lo que solucionaría los problemas. Mira:

´´´ ganancias_julio, gastos_julio = toma_valores_en_la_base ()

print (ganancias_julio / 100) print (gastos_julio / 100) ´´´

Y la respuesta: ´´´ 188.98 13.1 ´´´ ¡Genial! Con el código ordenado, lo compartí con las sucursales internacionales de mi empresa. Sin embargo, rápidamente recibí varias quejas que mostraban errores en los cálculos. Pero ¿por qué?

La primera queja provino directamente de nuestra filial en Túnez. El programa mostraba cálculos que eran claramente incorrectos para ellos: la base de datos almacenaba **10000 milim** y el valor impreso era de **100** dinares tunecinos, cuando debería ser **10**. Esto se debe a que, en Túnez (y en varios otros países) la subunidad de moneda principal no proviene de la centésima (**1/100**) pero de la milésima **(1/1000) -1 dinar tunecino equivale a 1000 milim**.

Entonces, tenemos un gran problema potencial con la internacionalización del código. De hecho, incluso si trabajamos con una sola ubicación, el peligro permanece. En el mundo financiero, las subunidades cambian con el tiempo debido a la inflación y la deflación.

Si almacenamos números enteros, tendremos que migrar los valores almacenados cada vez que haya un cambio, lo que puede dificultar enormemente el mantenimiento de todo el sistema.

Lo ideal aún sería poder trabajar con la unidad monetaria principal, con números decimales, **pero con exactitud**. ¿Es posible?

Trabajar con precisión con el tipo `Decimal`

Debido a esta necesidad recurrente de lidiar con números no enteros exactos, la mayoría de los lenguajes de programación nos brinda tipos específicos para lidiar con esto.

En el caso de Java, por ejemplo, tenemos el BigDecimal. En Python, tenemos todo el módulo decimal y, más concretamente, el tipo **Decimal**. Importándolo, su uso es directo:

```
from decimal import Decimal

ganancias_julio = Decimal ('99.91') * 5
print (ganancias_julio)

gastos_julio = Decimal ('110.1') * 3
print (gastos_julio)
```

Ahora, mira el resultado:

```
499.55
330.3
```

¡Exactamente como en la calculadora! Los analistas de la empresa ya no tendrán ningún problema más con los cálculos.

Usando la precisión exacta cuando la necesitamos

Empezamos con un problema tipo **float** de Python, no conseguía darnos un resultado exacto de un cálculo. Pronto comprendimos que el problema no era Python en sí, sino el float y cómo lo maneja la computadora.

Como se trataba de dinero, la precisión en los cálculos era fundamental. Entonces, necesitábamos alguna solución. Echamos un vistazo a cómo podemos convertir todo en números enteros (por ejemplo, convertir el valor monetario de Dólares a centavos), lo que a veces puede ser una buena salida.

Sin embargo, tratar todos los números como enteros tiene sus consecuencias negativas, como posible overflow y problemas de mantenimiento de código. Queríamos una solución mejor y ... ¡lo logramos!

Aprendimos que la mayoría de los lenguajes de programación tiene algún tipo numérico exacto para evitar este problema. En el caso de Python, este tipo es el**Decimal**, con **precisión arbitraria**. Con él, nuestros cálculos ganaron la precisión necesaria y terminamos con todo el problema inicial.

Una intuición natural después de conocer los tipos numéricos exactos en lenguajes de programación, como el **Decimal**, es querer usarlos para todo. A pesar de esto, es importante analizar siempre si vale la pena - los tipos exactos exigen más tiempo de procesamiento.

Por lo general, un pequeño error en el décimo decimal de un número es irrelevante y la precisión exacta no es necesaria. Por lo tanto, siempre tenemos que analizar el contexto de nuestro propio programa antes de aplicar una decisión.

¿Ya conocía el tipo Decimal? ¿Y toda esta confusión con el float? ¿Qué tal aprender más sobre **Python** y sus diversos recursos? Entonces, ¡Mira nuestros cursos de **Python para Data Science** aquí en Alura!

ARTÍCULOS DE TECNOLOGÍA > DATA SCIENCE

un solo pago de US$49,90

- 218 cursos
- Videos y actividades 100% en Español
- Certificado de participación
- Estudia las 24 horas, los 7 días de la semana
- Foro y comunidad exclusiva para resolver tus dudas
- Acceso a todo el contenido de la plataforma por 6 meses

¡QUIERO EMPEZAR A ESTUDIAR!

Paga en moneda local en los siguientes países

ANUAL

US$79,90

un solo pago de US$79,90